cc-69-25

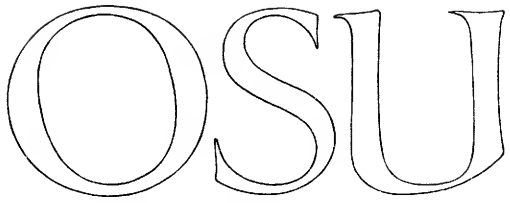
A Brief Description of OSCAR
(Third Revision)
(Describes Version 56)

by

Joel Davis

and

Gilbert A. Bachelor



COMPUTER CENTER

Oregon State University
Corvallis, Oregon 97331

A Brief Description of OSCAR

(Third Revision)

(Describes Version 56)

by
Joel Davis
and
Gilbert A. Bachelor

cc-69-25

September, 1969

INTRODUCTION

OSCAR (Oregon State Conversational Aid to Research) is an arithmetic interpreter for use at remote teletype or CRT connections to the CDC 3300. OSCAR can function in two ways. It can act like a sophisticated desk calculator executing direct statements one by one, or it can store programming away in blocks, then execute them (see sections M-P).

In the following, a box, [ ] , indicates a specially labelled key on the Teletype unit. [Control shift A] means hold the control shift key down while typing the character A. The control shift is above the upper case shift key. Characters printed by the computer are underlined in the examples below. Section I describes the system's operations, and the remaining section describes OSCAR.

I. Starting, Stopping, and Interrupting OSCAR.

OSCAR operates under the OS-3 system. The following are system operations needed to start and stop OSCAR. The user's identification is a positive integer of 5 digits or less (or a 4 letter word).

To start:

[Control shift A]

#Job number, user identification [carriage return]

#OSCAR [carriage return] OR # ↑

OSCAR AT YOUR SERVICE

To stop when done:

[Control shift A]

#LOGOFF [carriage return]

To interrupt OSCAR if it is calculating:

[Control shift A]

#MI [carriage return]

To interrupt OSCAR if it is typing:

[Break] [break release] [control shift A]

#MI [carriage return]

II. OSCAR

OSCAR reads statements, one line at a time, and obeys them immediately. A bell is rung when OSCAR has completed a line. A line must end with a [carriage return] (or sometimes [escape] or [alt mode], see below). A line starting with * or ending with * [carriage return] is ignored, as is any character followed by \. Statements are made up of constants, variables, and special words.

CONSTANTS:

These are numbers, arrays, and alphabetic strings:

1. Exact real numbers: 0; 1; -1; +1; 864216; 123; -12345678901234567890123456789012345678901234567890; 1.264P (P means precisely); 12E11 (means 12 X 10^{11}).

2. Inexact real numbers: 1.264314 (precision 7 decimal digits); -29.6E-6P18 (-29.6 X 10^{-6} with precision 18 decimal digits); -29.6D-6P18 (same as preceding example); 123. (When no P is given, a precision of at least 6 is assumed); 123.6P4 (precision 4); 1.23456789012345678901234567890123456789012345678901234567890 (precision 50 is maximum); 123.6E6E10 (123.6 X 10^{16} in precision 6); 123.4567P6 (precision 6 is used).

3. Rational numbers: 0; 1/2; -3/4 (in general, exact number/exact number).

4. Complex numbers: 1I; -2I; 1-3I; -6I; 1 + 29.6D-6P18I; 1/3 + 2I/7; 2.-3J (J may be substituted for I).

5. Vector arrays: ARRAY (1, 6, -3, 6/7, 1-3I) (A vector whose components are 1, 6, -3, 6/7, 1-3I).

6. Matrix arrays: ARRAY ((1, -3, 5) (2, 6, 1+4I) (I, -7, 8)) (A 3 by 3 matrix).

7. Alphabetic string: "ANY OLD MESSAGE"; "A = " (used in print statements); "THIS ENDS IN CARRIAGE RETURN; CR means "[carriage return]".

8. Logical: TRUE, FALSE (used as logical operands).

VARIABLES:

These are places in the 3300 memory to store the value of constants or other expressions for later use. A user refers to these by a 1 to 4 character name such as: A, B, M, X, TEMP, T1, T2, KEEP, MATRIX (only the first three letters plus the last letter are used), or any other convenient mnemonic. Some variables have a constant already in them. For example, PI is preset to 3.1415926535898, E is preset to 2.7182818284590, I and J to 1I. No type declarations are used. If a variable contains an array, the variable name can be subscripted by a non-negative constant, variable or expression. For example: V(1) (means V_1); V(M) (means V_m); M(1,2) (means M_{12}).

EXPRESSIONS:

These include any reasonable algebraic or arithmetic combination of constants, variable names, and operations (such as +, -, ...) which has a value.

Examples:

1.6; X; 1+2-(PI/2+180.*X)/2; -TEMP; A*B (means A times B); A↑B (or A**B, means A^B); -A*B; A*B+C*D (means (A*B)+(C*D)); A>B (has value TRUE or FALSE); A OR B (A and B must be TRUE or FALSE. OSCAR also recognizes AND, XOR, and NOT); "AB" + "C" (gives "ABC" as a value); A + B (A and B could refer to real or complex arrays, or to alphabetic strings); A*B (could mean matrix multiplication or matrix times an array); LOG X + SIN(X + Y ↑ 2) + EXP 2; V(1:6) (means positions 1-6 of V).

STATEMENTS:

These can be used as direct and immediate commands to OSCAR. Several statements, separated by semicolons, can be put in one line; in this case, none are executed until the line is completed with carriage return.

A. Replacement Statement. Examples:

X←6 (the original contents (if any) of X are ignored; the constant 6 is placed in X).

V(2)←X + 6. (If necessary, V is made into an array large enough to hold V_2. V_2 is set equal to the value of (X + 6));

V = 6 (the equal sign, =, can be used for ← in the simple replacement statement, which has the general form:

variable name ← expression).

B. IS Statement. There is a key labelled escape (or sometimes alt mode) which causes no printing or carriage action. This is used in IS statements, for example:

2 + 4 IS escape 6 (the machine types 6, any expression can be on left)

Y ← 2 - 7 + (X + Y)/2 IS escape 68.264 (Y is set to 68.264 also).

C. IF Statement. Examples:

IF X < 6 THEN Y ← 6 ELSE Y ← X ← 1;

IF -X*Y NEQ 29 THEN [X ← 6; Y ← 7] ELSE X ← Y;

IF X=Y THEN X ← 8 (otherwise do nothing);

IF S THEN X ← 7 ELSE Y ← 7 (S must be TRUE or FALSE).

D. FOR Statement. Examples:

FOR I = 10 TO 60 DO A (I) = I (means A(10) ← 10; A(11) ← 11; ...; A(60) ← 60);

FOR J = 10 BY -1 TO 1 DO [A(J) ← J; B(J+1) ← J ↑ 2];

FOR X = 0 BY .1 TO 1.0 DO ABV (ABV contains a literal expression, see Part J following).

E. WHILE Statement. Examples:

WHILE A(I) GEQ 7 DO I ← I + 1 (means keep doing I ← I + 1 until A(I) is not greater than or equal to 7. OSCAR recognizes <, >, =, EQ, GEQ, NEQ, LEQ);

WHILE TRUE DO ABV (means do ABV repeatedly, ABV must contain a literal expression, see Part J, following).

WHILE X<7 DO PART 1 (see Part M).

F. READ Statement. Examples:

READ X,Y (will cause OSCAR to expect a list of two constants to be typed by the user, each constant terminated by a comma or carriage return. The first constant is put in X, the second into Y);

READ "X=",X (will cause OSCAR to print X = then await the user's typing of a constant to be entered into X).

G. PRINT Statement. Example:

PRINT "A=", A, CR, "B=", B, CR

A = 6.24932
B = 67.2345

(These would be printed by OSCAR.)

PRINT A+B↑2+(X+Y)/2,A(I),6 (Expressions can be printed.)

H. FUNCTION or SUBROUTINE DEFINITION Statement. Examples:

LET F(X) = X*X (Then, for example, F(2) IS 4);

LET F2(X,Y,Z) = X + Y + Z↑2 (Thus, F2(6,3,2) is 13);

LET SR(X,Y,Z) = X + Y + F(Z) (Same as F2);

LET F4(X,Y,Z) = [TEMP = Z ↑ 2; X + Y + TEMP] (Same as F2);

LET F5(X,Y) = [ABV; X + Y] (ABV contains a literal expression, see Part J);

LET G (N) = IF N < 2 THEN 1 ELSE N*G(N-1) (G(N) is N!);

LET P(X,Y) BE PRINT X,Y,CR (Defines a subroutine to print X, Y, and CR. P(X,Y) has no value.)

LET P2(X,Y) BE [PRINT X,Y,CR; X ← Y ← 0]

I. SUBROUTINE CALL Statement. Example:

P(6,Z-56.78) (If P were defined as in Part H, this statement would cause 6, Z-56.78, and CR to be printed.)

J. ABBREVIATION Statement. An expression enclosed in apostrophes is called a literal expression. Literal expressions (or variables containing them) can be used wherever expressions are allowed. Examples:

X:: = 'A + D + C/3' (means make X an abbreviation for the expression (A + D + C/3). Wherever X is typed

in any statement of type A through I, OSCAR uses (A + D + C/3) instead);

ABV:: = 'X ← X ↑ 2; Y ← Y ↑ 3' (Used, for example, in the definition of F5 in Part H);

X:: = 'Y'; X ← 6 (This assigns 6 to Y.)

K. CLEAR Statement. Example:

CLEAR X,F5 (The variables named X and F5 are cleared or made empty. Primarily used to clear procedures or abbreviations.)

L. Commands. These are used to manipulate files, to alter OSCAR, to print internal structures of OSCAR, and miscellaneous other purposes. A command must be the first and only statement on a line and has a & as its first character. A complete list of such operators is included in the appendix. Examples:

&REWIND,26 (This rewinds file 26)

&INPUT,14 (Further READ statements will read data from file 14.)

&INPUT,14,TTY (This is the same as above, except, as each line is read from 14, it is displayed on the Teletype.)

&INPUT,TTY (Further READ statements will read from the Teletype.)

&OUTPUT,15 (The output of further PRINT or IS statements goes to file 15.)

&OUTPUT,15,TTY

&OUTPUT,TTY

&CONTROL,8 (Direct statements are read and executed from file 8; on reaching end of file or end of data, control is returned to the Teletype.)

&CONTROL,8,TTY (This is the same as above, except that before the execution of each line, the line is displayed on the Teletype.)

&PRECISION =10 (This changes the standard precision to 10; so, for example, 1.6 now means 1.6P10 instead of 1.6P6).

&OCTOUT A,B,C (The internal form of the variables A,B and C is printed in octal).

&DATE (The current date and time are printed).

M. Store Statements. Examples:

12.34: A ← X + Y (The text A ← X + Y is stored in part 12, as step 12.34. The statement A ← X + Y is neither examined nor executed. The previous contents (if any) of step 12.34 are lost. All the steps of part 12 must start with 12., the decimal fraction is arbitrary (up to six decimal digits).

10.21:&INPUT,21 (Any sort of statement may be stored including all of those from section A through section P).

N. DO PART or DO STEP Statements. Examples:

DO PART 12 (Means do the steps in part 12, starting with the numerically smallest; and proceeding to steps with ascending step numbers until the end of a part is reached. If this last statement is <u>not</u> a GO TO statement the DO PART statement is complete, otherwise the GO TO statement is executed and the search for the end of a part is continued. A GO TO statement (see section O) may change the above order of execution of the steps or even the part being done.)

DO PART 12.61 (This is the same as above, except execution is started at the first step in part 12 numbered 12.61 or higher).

DO STEP 12.61 (Means do the one statement at 12.61).

DO PART X+Y-1 (The value of X+Y-1 is used as the parameter of the DO PART statement).

DO STEP V(I)

Note: The DO PART and DO STEP statements can be used as direct or stored statements or in the body of FOR, FUNCTION, IF, WHILE, or ABBREVIATION statements.

O. GO TO Statements: Example of use in a stored program:

8.1: READ X

8.2: PRINT X↑2, X↑3, CR

8.3: GO TO 8.1

DO PART 8 (This will produce an infinite repetition of the following:

Read a number, then print its square and cube.

Typing MI (standing for manual interrupt) as data for the read statement will terminate this program).

9.2: GO TO X+Z (The value of (X+Z) is used as the parameter in this GO TO statement).

Note: A GO TO statement can be used in stored programming only.

P. Editing Statements. Examples:

Suppose the program defined in section O as part 8 has already been stored.

8.25: PRINT X↑4, X↑5, CR (This is a store statement that would effectively insert a line between 8.2 and 8.3).

ERASE PART 8 (Would eliminate part 8).

ERASE STEP 8.1 (Would eliminate step 8.1).

PRINT PART 8 (Would print all the steps of part 8, including step numbers).

PRINT STEP 8.2 (Would print the step 8.2 including its step number).

&PROGRAM,5,.01 (Would print 5.01: and expect the user to type in the contents of step 5.01. Then 5.02: would be printed by OSCAR, with its contents typed by the user. This would continue indefinitely until escape is pushed.)

USER-DEFINED OPERATORS:

The OSCAR user can define new operators of his own choosing. To do this, the user first picks a name for the operator. The name must start with a question mark (?), and may include up to 3 letters and/or digits after the (?). For example, one could use the following as names for operators:

? ?A ?3X ?ADD ?42

An operator is defined by a LET statement, and must have two arguments. For example:

LET ?Q(X,Y) = SQRT(X↑2 + Y↑2)

An operator so defined is then used between its operands, like other operators. For example, B:=3 ?Q 4 will assign the value 5 to B. The statement C:=3 ?Q 4 ?Q 12 will assign the value 13 to C. (3 ?Q 4 is 5, and 5 ?Q 12 yields 13).
All ?operators are considered to have the same precedence, which is higher than all other operators except %.

NEW FEATURES:

1) Comments are now allowed in stored program steps. For example, 1.05:*COMPUTE THE SUM is now legal. Executing such a step does nothing; the value of the step is [] (undefined).

2) OSCAR can now be used in a batch job. If OSCAR is called by the statement 8OSCAR, it will treat unit 60 (the card reader) as its "teletype" for input, and unit 61 (the line printer) as its "teletype" output unit. All statements, commands and data read from the input unit will be printed on the output unit, with a mode indicator at left. Outputs from OSCAR to its "teletype" will be printed on the output unit, with no mode indicator. A record read from the input unit that contains only the word ESCAPE, starting in column 1, will cause OSCAR to behave just as it does when a teletype user presses the ESCAPE key (at the beginning of a line). (ESCAPE changes modes or interrupts data input).

If OSCAR is called by a statement of the form 7OSCAR,I=(lun), then OSCAR will use the specified logical unit number as its "teletype" input unit and will otherwise behave as described above. The input unit must already be equipped before OSCAR is called.

A file mark (or end of file card) on the input unit denotes the end of input for OSCAR, and it will print END OF OSCAR RUN and return to control mode. If any error occurs and the user is not using DEBUG (see next section), OSCAR will abort its run with the message **ABNORMAL END OF OSCAR RUN. (This is printed after the error messages.)

3) The DEBUG command is not new but it has a new feature for the convenience of batch users of OSCAR. If a batch user of OSCAR does not want the run aborted in case of an error, he should include the command &DEBUG ON or the command &DEBUG ON (lun). (The & is printed as a $\leq$ on the line printer.) If the command DEBUG ON has been used, then a subsequent error will produce a special message of the form ERROR AT (addr) (message), a few blank lines, then the normal error messages and OSCAR will go on reading from its input unit instead of quitting. If the command DEBUG ON (lun) has been used, then an error will produce the special message mentioned above. OSCAR will then read commands (commands only) from the logical unit specified in the DEBUG ON statement. Each command read will be printed and then executed. This will continue until the command GO is read or a record is read containing only the word ESCAPE. OSCAR will then print the regular error messages and go on reading from its input unit.

Teletype and CRT users can also use the DEBUG feature, with the additional provision the OSCAR will read commands from the console if an error occurs after a DEBUG ON command has been given. An "escape" will terminate this special command mode. The DEBUG ON (lun) works just as described above.

To cancel the debug mode, use the command DEBUG OFF.

4) When OSCAR is reading from a file (or a card reader, in a batch job) it will recognize a line containing only the word ESCAPE. If OSCAR is reading statements or commands from the file, ESCAPE causes it to switch from normal to command mode or vice versa. If OSCAR is reading data from the file, ESCAPE causes an "interrupt" just as if MI had occurred. Whenever the &CONTROL, (lun) command is given OSCAR will start reading from the (lun) in normal mode. At the end of a file, OSCAR switches back to the standard input unit (console, card reader, etc.) and also reverts to normal mode.

5) For CRT users, there is now a RECORD feature, which enables the user to get a "hard copy" of everything he does in OSCAR. In conjunction with this is a "recall" feature, which enables the CRT user to bring back to the screen a display of what he has done earlier.

The RECORD feature can be activated in either of two ways. The easiest way is to call OSCAR with a statement of the form ≠OSCAR,R=(lun). This will cause OSCAR to record on the specified (lun) and this (lun) will be equipped as a file if it is not already equipped. The other way to start recording is to use the command ≤RECORD ON (lun). This will cause OSCAR to record on the specified (lun), which must have been equipped previously. The record feature can be turned off by the command ≤RECORD OFF. While the Record feature is on, OSCAR displays an "R" next to the second mode indicator at the right side of the screen.

When "Record" is on, every line displayed on the screen, whether it is an input by the user or an output from OSCAR, will be written on the specified record unit. The first two characters of each recorded line are blanks and the lines are from one to thirteen words long (4 to 52 characters). (Trailing blanks are removed.) This output can thus be written directly on a line printer or can be written on a file and later copied to a line printer, if desired.

The "recall" feature mentioned above is available only when the Record feature is in use. Also, the record unit must be a

file (since one cannot backspace a line printer). If one is recording on a file and if one is in normal mode ("blotch" displayed in corner), then one can "recall" anything which has been recorded on the file. To do this, one types a special "command" whose first character is the "blotch". The SKIP key is used to space past the blotch without wiping it out. Then one types either a decimal integer or a minus sign followed by a decimal integer and presses SEND. If an unsigned integer was typed, OSCAR will rewind the Record unit then space forward as many records as specified by the integer. It will then read the next sixteen records from the file and show them on the screen. Finally, it searches forward to the end of data on the file, to be ready for further recording, etc.

If a negative integer was typed, OSCAR backspaces the file the number of records specified by the integer. It then acts as described above (reads forward sixteen records, shows them on the screen, etc.).

If, for any reason, the recall command cannot be carried out, OSCAR does nothing. If the file is positioned to a point less than sixteen records from its end, then fewer than sixteen lines will be shown on the screen. (Note: the recalled information is not recorded on the file again.)

One convenient use of the recall feature is to position a previously typed statement near the top of the screen, carriage return down to it and read it back into OSCAR, possibly after making changes in it.

CHARACTER STRING PROCESSING

Previous versions of OSCAR had some ability to process character strings (add, subtract, compare). Version 55 has several new features which make it possible to do general character string manipulations. These features include subscripting and sub-array notation to refer to parts of character strings, a search feature and DECODE and ENCODE functions. To indicate the possibilities, three simple applications which have been

programmed for OSCAR V55 are a "form letter" processor, a Markov Normal Algorithm processor and a simple expression translator (from algebraic expressions to Polish strings). We shall describe OSCAR's character string processing features (both old and new) on the following pages.

INPUT, OUTPUT AND REPRESENTATION OF CHARACTER STRINGS

The internal representation of character strings in OSCAR has been changed. In previous versions, a string was terminated by a quote mark ("). This meant that a quote mark could not be included in a string. The new representation includes an integer (internally) that tells how many characters the string contains. This makes it possible to include any characters in a string.

There are two ways of representing a character string in OSCAR language. One way is to enclose it in quotation marks, as for example: "THIS IS A STRING". This notation is fine for teletype users. There is no quotation mark on the CRT keyboard; however, OSCAR, when used at a CRT, allows the notation ≠≠ to represent a quotation mark. (On output to the CRT, a quotation mark is displayed as a blotch.) Unfortunately, EDIT does not recognize the ≠≠ notation and treats it as two apostrophes in a row (''). A CRT user who wants to use EDIT to prepare or to modify an OSCAR program cannot use the quotation mark at all. To alleviate this problem, there is another notation for character strings: TEXT /THIS IS A STRING/ means the same as the previous example. In this second notation, the word TEXT can be followed by any non-space character (for example: / + : $, etc.), then the string of characters and finally, the chosen "bracket" character again. Thus, TEXT/X=/, TEXT:X=: and TEXT%X=% all represent the same character string "X=".

There are three special conventions regarding the contents of character strings. These are: 1) The bracketing character (" or whatever bracketing character is used with TEXT) can be included within a character string by writing it twice in a row.

For example, "X""Y" represents the character string X"Y. Other examples; TEXT =X=== represents the string X=, """" represents the string ", and "" represents the empty string. 2) Carriage return (end of line) always terminates a character string and the carriage return is included in the string as its last character. Thus, if PRINT "RESULTS ARE: is the last thing on a line, the character string to be printed will be RESULTS ARE:, followed by a carriage return. 3) The character @ (↓ for CRT users) denotes a carriage return within a character string. For example:

"FIRST LINE@2ND LINE@LASTLINE@"

The three @ symbols in this character string represent carriage returns. As a result of this convention, it is not possible to type in a character string that actually has the character @ (or ↓) in it. The only way to get a character string with @ in it is to use the ENCODE function (see later paragraph).

When a character string is printed (using PRINT), the characters in the string are simply typed out, with no bracketing characters, and carriage returns cause output to go to the next line. If EPRINT is used, the bracketing characters " " are printed; quotation marks within the string are printed twice, and carriage returns are printed as @. The result is that an EPRINTed character string can be read back in and will produce a character string identical to the one that was EPRINTed (unless it contained a @ character; such a character will go back in as a carriage return). The constant CR is a special case; this represents a one-character string containing a carriage return but it will be printed as a carriage return whether PRINT or EPRINT is used.

There are three ways in which character strings can be read in by OSCAR.

1) A statement such as READ X will read the next constant from the input unit and assign it to X. The constant can be a number, array, etc., or a character string, of either the "..." form or the TEXT/.../ form.

2) The statement READLINE X will read an entire line (or the rest of a partially read line), put it into a character string and assign it to X. In this case, the carriage return at the end of line will not be included in the character string.

3) The statement READCHAR X will read the next single character from the input unit and assign it to X as a one-character string. At the end of a **line**, READCHAR will get an empty character string instead of a string containing a carriage return.

The statements READ, READLINE and READCHAR may each include a list of variables to be read. For example, READLINE X, Y, Z will read three lines and assign them to the variables X, Y and Z.

OPERATIONS ON CHARACTER STRINGS

Character strings can be "added", "subtracted" or "compared".

X+Y If X and Y are character strings, X+Y is a character string composed of X followed by Y (concatenation). For example, "STREET" +"CAR" is the string "STREETCAR".

X-Y If X and Y are character strings, X-Y is found by searching X to see if Y occurs within it (as a substring). If so, the part of X that matches Y is removed, and the value of X-Y is what is left. (X and Y are not changed.) If Y is not found in X, the value of X-Y is simply X. Another way of expressing X-Y is SAR(X,Y,""), that is, search and replace by empty (see next section for SAR function). For example, "CHARACTER"-"AC" is "CHARTER". Another example: "WALLA WALLA"-"ALL" is "WA WALLA" (only the first occurrence of "ALL" is removed).

X EQ Y If X and Y are character strings, X and Y are compared character by character to **see** if they are the same. If X and Y are the same length, and all corresponding characters match, then the value of X EQ Y is TRUE; otherwise, the value is FALSE.

X NEQ Y This is like X EQ Y except that it gives the result TRUE if X is not the same as Y; if X and Y are the same, the value is FALSE.

X<Y If X and Y are character strings, then corresponding characters of X and Y are compared, starting at the left, until two corresponding characters are different or the end of one (or both) of the strings is reached. If the strings are of the same length and all characters of X match the corresponding characters of Y, the value of X<Y is FALSE. If X is shorter than Y, but X matches Y as far as it goes, the value of X<Y is TRUE. If Y is shorter than X, and matches X as far as it goes, then X<Y is FALSE. If there is some position where a character in X does not match the corresponding character of Y, then X<Y is TRUE if the internal code for the character in X is less than the code for the character in Y; otherwise, X<Y is FALSE. (In comparing codes, letters and digits are considered to be "greater" than other characters.) (See table of codes at the end of this report.

X GEQ Y This is the same as X<Y except that it gives the answer TRUE if X<Y is FALSE, and vice versa.

X>Y This is the same as Y<X.

X LEQ Y This is the same as Y GEQ X.

FUNCTIONS WHICH CAN BE USED WITH CHARACTER STRINGS

Of course, the OSCAR user can define functions of his own which take character strings as arguments or produce character strings as values, or both. However, there are several "predefined" functions available in OSCAR which can be used with character strings. These are:

DECODE If X is a character string, then DECODE(X) is an integer in the range 0 to 63, giving the internal (OSCAR) code for the first character of the string. If X is an empty character string, then DECODE(X) has the value -1.

ENCODE If N is an integer in the range 0 to 63, then ENCODE(N) is a one-character string containing the character whose internal code is the specified integer.

SRCH If X and Y are character strings, then SRCH(X,Y) is an integer telling where string Y occurs within X, if it does occur. If it does not occur, the value of SRCH is zero. For example, SRCH ("ABCDE","CD") has the value 3. (CD occurs within ABCDE, starting at the third character of ABCDE.) If Y occurs in X in more than one place, SRCH indicates the location of the left-most occurrence of Y.

SAR If X, Y and Z are character strings, the value of SAR(X,Y,Z) is a character string formed in the following manner: X is searched for an occurrence of Y (using SRCH); if found, a new string is constructed in which the part of X that matched Y is replaced by Z. This new string is the value of SAR. If Y does not occur within X, the value of SAR is simply X. In any case, X, Y and Z are not changed. For example, SAR("OSCAR","CAR","WALD") has the value "OSWALD".

SIZE If X is a character string, SIZE(X) is an integer (0 or larger) telling how many characters the string X contains.

KIND KIND(X) produces a character string telling what kind of value X has.

Note: The arguments of the functions described above can be any expressions, so long as the expressions have the proper kinds of values.

SUBSCRIPTING AND SUB-ARRAYS

Subscript notation can be used to refer to individual characters in a string and the sub-array notation can be used to refer to sub-strings of a string.

X(I) If X is a character string and I is an integer in the range $1 \leq I \leq$ SIZE(X), then X(I) is a one-character string containing just the Ith character of X. If I = 0 or if I > SIZE(X), then X(I) is an empty character string. For example, if X = "ABCDE", then X(4) is "D" and X(6) is "".

X(I:J) If X is a character string and I and J are integers such that $1 \leq I \leq J \leq$ SIZE(X), then X(I:J) is a character string containing the Ith through Jth characters of X, inclusive. If any of the three inequalities specified above is not true, then X(I:J) is an empty character string. For example, if X = "ABCDE", then X(2:4) is "BCD", X(4:6) is "", and X(4:1) is "".

Note: One cannot assign values to x(I) or to X(I:J), when X is a character string. For example, X(3):="Q" is illegal if X is a character string. One can accomplish the effect desired in this example by the statement

X:= X(1:2) + "Q" + X(4: SIZE(X))

OTHER NOTES ABOUT USE OF CHARACTER STRINGS

As we have implied in the foregoing paragraphs, variables may have character strings as their values. This can happen either by reading in a character string or by an assignment statement (such as the example above). One can use either ordinary assignment (:=) or the old value assignment (:==) in dealing with character strings. One can also use the exchange operator (==).

Another useful concept to keep in mind, is that the elements of an array can be character strings. For example, A(1):="THIS"; A(2):="IS"; A(3):="AN"; A(4):="EXAMPLE." If the array A has only these four elements, then PRINT A will produce the sentence THIS IS AN EXAMPLE. Also, A(4,1:4) is the string "EXAM". (This selects the first through fourth characters of the fourth element of A.) Another way to define an array containing character strings is to use an ARRAY constant. For example, B:=ARRAY("THIS", "IS", TEXT/ANOTHER/,"EXAMPLE.") (One can use either the "..." or the TEXT notation in an ARRAY constant.) If B is PRINTed, the result is THIS IS ANOTHER EXAMPLE. And finally, PRINT B-A would produce OTHER. (Actually, B-A is an array whose Ith element is B(I)-A(I), for I=1,2,3,4. But, three of these elements are empty character strings; only the third one actually contains any characters.)

OSCAR internal character codes

The following table gives the character codes used in OSCAR. These codes determine the values of the DECODE and ENCODE functions, and also determine the results of comparisons such as <, GEQ, etc. (TTY denotes the teletype character; CRT denotes the display console character; and LP denotes the Line Printer character.)

OSCAR code	card code	TTY char	CRT char	LP char	OSCAR code	card code	TTY char	CRT char	LP char
0	0	0	0	0	32	0,6	W	W	W
1	1	1	1	1	33	0,7	X	X	X
2	2	2	2	2	34	0,8	Y	Y	Y
3	3	3	3	3	35	0,9	Z	Z	Z
4	4	4	4	4	36	12,3,8	.	.	.
5	5	5	5	5	37	11,6,8	@	↓	↓
6	6	6	6	6	38	5,8	&	≤	≤
7	7	7	7	7	39	11,0	!	∨	∨
8	8	8	8	8	40	(none)	carriage return		
9	9	9	9	9	41	0,7,8	?	∧	∧
10	12,1	A	A	A	42	12,5,8	#	≥	≥
11	12,2	B	B	B	43	4,8	'	≠	≠
12	12,3	C	C	C	44	12,6,8	"	■	¬
13	12,4	D	D	D	45	12,0	<	<	<
14	12,5	E	E	E	46	3,8	=	=	=
15	12,6	F	F	F	47	11,7,8	>	>	>
16	12,7	G	G	G	48	12	+	+	+
17	12,8	H	H	H	49	11	-	-	-
18	12,9	I	I	I	50	11,4,8	*	*	*
19	11,1	J	J	J	51	0,1	/	/	/
20	11,2	K	K	K	52	11,5,8	↑	↑	↑
21	11,3	L	L	L	53	0,6,8	←	'	≡
22	11,4	M	M	M	54	0,4,8	(	(	(
23	11,5	N	N	N	55	12,4,8	)	)	)
24	11,6	O	O	O	56	7,8	[	[	[
25	11,7	P	P	P	57	0,2,8	]	]	]
26	11,8	Q	Q	Q	58	2,8	:	:	:
27	11,9	R	R	R	59	0,3,8	,	,	,
28	0,2	S	S	S	60	12,7,8	;	▲	;
29	0,3	T	T	T	61	blank	space		
30	0,4	U	U	U	62	11,3,8	$	$	$
31	0,5	V	V	V	63	6,8	%	%	%

APPENDICES TO OSCAR MANUAL

(It is planned that, eventually, there will be a manual for OSCAR. This manual has not yet been written, but there is a need for information about OSCAR. Hence, we present here some Appendices for the manual. These provide lists of reserved words, special symbols, etc.)

A. List of Reserved Words.

B. List of Special Symbols.

C. Order of Precedence of Operations.

D. Constants and Data Inputs.

E. Forms of Statements.

F. List of Commands.

APPENDIX A: Reserved Words and Special Variables

The following is a list of reserved words in OSCAR. These will be discussed on succeeding pages. Words preceded by an asterisk (*) have not been implemented yet. Use of any word of this list in other than its intended use may cause diagnostic messages.

ABS
ACCEPT
* ALL
AND
ARCCOS
ARCSIN
ARCTAN
ARG
ARRAY
BE
BY
CLEAR
COS
CR
DECODE
DENOM
DET
DIV
DO
DONE
ELSE
ENCODE
ENTIER
EPART
EPRINT
ERASE
EQ
EXIT
EXP
EXPR
FALSE
FOR
FP
GCD
GEQ
GO
GOTO
GTR
IF
IM
IP
IS
KIND
LEQ
LET
LN
LOG
LOGT
LSS
MAX
MIN
MOD
NEQ
NOT
NPART
NUM
OR
PART,PARTS
PDL
POP
PREC
PRINT
PROCEDURE
* PRODUCT
PUSH
RE
READ
READCHAR
READLINE
SAR
SET
* SIGMA
SIGN
SIN
SIZE
SQRT
SRCH
STEP,STEPS
STOP
TAN
TEXT
THEN
TO
TRUE
TYPE
UNTIL
VALUE
WHILE
WRITE
XOR
ZARRAY

The following is a list of names of special variables in OSCAR. These are ordinary variables in the sense that they can be used for any purpose by the user. However, these have initial values when OSCAR is called, and can be used by the user without his defining them.

Special Variable	Initial Value
E	2.7182818284590
I	1I ($\sqrt{-1}$)
J	1J ($\sqrt{-1}$)
O (letter)	0 (zero)
PI	3.1415926535898

The special variable INDEX has no initial value, but whenever SAR is used, INDEX is set to an integer value telling where the replacement was made. (Zero means no replacement occurred.)

List of reserved words and special variables with brief description.

ABS	Absolute value function.
ACCEPT	Same as READ.
ALL	Reserved for future use.
AND	Logical operator.
ARCCOS	Arccos function.
ARCSIN	Arcsin function.
ARCTAN	Arctan function.
ARG	Arg (x,y) is the angle of the complex number x + y*1I. Arg (z) is the angle of the complex number z.
ARRAY	Defines array constant.
BE	Used in procedure definitions.
BY	Identifies increment value in FOR statement.
CLEAR	Clears variables (makes them undefined).
COS	Cosine function.
CR	Constant. Its value is "(carriage return)."
DECODE	Decode function. Argument is character string. Result is an integer (the OSCAR code for the first character of the string).
DENOM	Function, denominator of a rational number (see NUM).
DET	Determinant function.
DIV	Integer division operator.
DO	Do the expression or statement that follows.
DONE	In a stored program, indicates that we are done with this part (terminates execution of a DO PART).
E	Special variable. Initial value is 2.7182818284590.
ELSE	Follows true alternative, precedes false alternative, in conditional statement or expression.
ENCODE	Encode function. Argument is an integer. Result is a character string, the character whose OSCAR code is the given integer.
ENTIER	Entier function (greatest integer less than or equal to argument).
EPART	Function, exponent part of number (see NPART).
EPRINT	Emphatic print. Prints actual form of quantity (see PRINT).

ERASE	Erase step 2.6 erases step 2.6 from a stored program.
EQ	Relational operator, equal to.
EXIT	EXIT means exit from a FOR statement or a procedure. EXIT TO (step number) means EXIT and then GO TO the specified step.
EXP	Exponential function (e to the power).
EXPR	If Y is a variable which represents a literal expression, X←EXPR(Y) will copy the expression to X. (X←Y would evaluate the expression).
FALSE	Logical constant.
FOR	Beginning of a FOR statement.
FP	Function, fraction part of a number (see IP).
GCD	Function, greatest common divisor of arguments.
GEQ	Relational operator, greater than or equal to.
GO	Same as GO TO.
GO TO	Transfer control to a new step (can be used in stored programming only).
GOTO	Same as GO TO.
GTR	Relational operator, greater than.
I	Special variable. Initial value is 1I (equivalent to 1J).
IF	Beginning of a conditional statement or expression.
IM	Function, imaginary part of a complex number (see RE).
INDEX	Special variable. See page A.2.
IP	Function, integer part of a number (see FP).
IS	Print value of preceding expression.
J	Special variable. Initial value is 1J (equivalent to 1I).
KIND	Function. Value is a word indicating what kind of quantity the argument is.
LEQ	Relational operator, less than or equal to.
LET	Beginning of a function or procedure definition.
LN	Natural logarithm function.
LOG	Same as LN.
LOGT	Function, logarithm to base ten.
LSS	Relational operator, less than.
MAX	Function, maximum of arguments.
MIN	Function, minimum of arguments.
MOD	Modulus operator, remainder of division.
NEQ	Relational operator, not equal to.
NOT	Unary logical operator.

NPART	Function, number part of number (so X=NPART(X)*10↑(EPART(X)).
NUM	Function, numerator of a rational number (see DENOM).
O (letter)	Special variable, initial value is 0 (zero).
OR	Logical operator, inclusive or.
PART	A section of stored program.
PARTS	Same as PART.
PDL	Defines push-down list constant.
PI	Special variable. Initial value is 3.1415926535898.
POP	Pop up each variable following (see PUSH).
PREC	Function, gives precision of argument.
PRINT	Print values of following expressions.
PROCEDURE	Same as LET.
PRODUCT	Reserved for future use.
PUSH	Push down each variable following (see POP).
RE	Function, real part of complex number (see IM).
READ	Read in values for variables following, print out values of constants. (Note: Character strings are constants.)
READCHAR	Reads one character as an alphabetic string.
READLINE	Reads an entire line as an alphabetic string.
SAR	Search and replace function.
SET	Beginning of an assignment statement (may be omitted in most situations).
SIGMA	Reserved for future use.
SIGN	Function, value is 1, 0, or -1 according as argument is positive, zero, or negative.
SIN	Sine function.
SIZE	Function, value is integer giving size of argument (number of elements in a vector, for example; see SIZE function table).
SQRT	Square root function.
SRCH	Search function.
STEP	Identifies increment value in a FOR statement. Also step refers to a step in a stored program.
STEPS	Same as STEP.
STOP	Stop execution of program and revert to normal mode.
TAN	Tangent function.
TEXT	Provides another representation for character strings: TEXT/string/.

THEN	Follows IF clause, precedes true alternative, in a conditional statement or expression.
TO	Identifies final value in a FOR statement (unless it follows the word GO).
TRUE	Logical constant.
TYPE	Same as PRINT.
UNTIL	Identifies final value in a FOR statement
VALUE	Function, value is value of argument (not very useful).
WHILE	Beginning of a WHILE statement.
WRITE	Same as PRINT.
XOR	Logical operator, exclusive or.
ZARRAY	Defines array constant with subscript starting at zero.

SIZE FUNCTION TABLE

Kind of Argument	Value
Logical, inexact, exact but not rational	Number of machine words used in number
Character string	Number of characters
Rational or complex	2
Array: vector	Number of elements
Array: matrix	Number of rows

Appendix B

Special symbols in OSCAR

.	Decimal point.
+	Addition operator (concatenation operator for character strings).
-	Subtraction operator (also character string deletion operator).
*	Multiplication operator.
/	Division operator.
↑	Exponentiation operator (raise to power).
**	Same as ↑ .
%	Precision operator. X % 12 has same value as X, but with precision 12.
<	Relational operator, less than (same as LSS).
=	Assignment operator (equivalent to ←), or relational operator (equivalent to EQ), depending on context.
>	Relational operator, greater than (same as GTR).
<=	Relational operator, less than or equal to (same as LEQ).
=<	Same as <= .
=>	Relational operator, greater than or equal to (same as GEQ).
>=	Same as => .
<>	Relational operator, not equal to (same as NEQ).
><	Same as <>.
←	Assignment operator (assign value to variable on left).
:=	Same as ← .
::=	Emphatic assignment operator (same as ← , except when the variable on left represents a literal expression, in which case, the variable is cleared before assigning new value).
←=	Old value assignment operator. (Same as ←, except value of the assignment statement is the old value of the left operand, not the new value).
←←	means ::=
=←	means ←=
==	Exchange operator (exchange values of variables).
:=:	Same as ==.
:	Colon operator. Defines number pair for selecting portion of an array. For example, X(3:6).
:==	Means ←=

, Comma. Separates subscripts in a subscripted variable. Separates variables or expressions in list following READ, PRINT, etc. Separates elements in an ARRAY constant.

; Semicolon. Separates statements. Also causes preceding statement to be executed.

(Left parenthesis.

) Right parenthesis.

[Left bracket (statement parenthesis).

] Right bracket (statement parenthesis).

[] An empty variable.

! Absolute value bracket (left or right, depending on context).

!expr! is equivalent to ABS (expr) .

' Literal expression bracket (left or right, depending on context).

" Character string bracket.

& Illegal, except in a character string, or unless it introduces a command (see part F).

@ Illegal except in a character string (denotes carriage return).

? First character in the name of a user-defined operator.

Illegal except in a character string.

$ Same as a semicolon (;) in an OSCAR statement. Denotes "bypass" on input to a READ statement.

\ Reverse slant denotes backup. (Cancel preceding character).

* A * at beginning or end of a line cancels the line.

... Three dots in a row denote continuation to a new (physical) line.

(space) Space separates items.

(return) Return denotes end of line.

(line feed) Line feed is ignored.

(escape) Escape denotes end of line, except at beginning of line, where it causes a change of mode or an interrupt.

(altmode) Same as (escape).

(control W) Control W is same as escape.

(control R) Control R means change to tape input mode.

(control T) Control T means change back to keyboard input mode.

MI denotes "Manual Interrupt" if read by a READ statement.

ESCAPE If a line read from a file contains only the word ESCAPE, this is treated the same as the (escape) key at a teletype.

Appendix C

Order of precedence of operations in OSCAR

Generally, execution of operations in an OSCAR expression, is left-to-right, with the exceptions discussed below.

(1) Expressions within brackets ( ) [ ] ! ! are evaluated before operations outside them. For example, in A*(B+C) , the addition is done first, then the multiplication.

(2) The evaluation of an expression within literal brackets (apostrophes) is postponed as long as possible, consistent with other rules. For example:

In A(2*I)+B(J-1) the order of operations is *, subscript A, -, subscript B, +.

In 'A(2*I)'+B(J-1) the order is -, subscript B, *, subscript A, +. (The evaluation of 'A(2*I)' is postponed until it actually has to be carried out).

In X←'A+B/C' the literal expression 'A+B/C' is assigned to X without evaluating it. Subsequent references to X cause the expression to be evaluated.

(3) Certain operations are done in right-to-left order. This includes assignment operations (←, :=, ::=) and evaluation of functions. For example:

In A←B←A+B, the value of A+B is assigned to B and then to A.

In SQRT ABS X , the absolute value of X is computed, and the square root of this result is found.

In A(I)←B(J)←S+T, first, A is subscripted, then B is subscripted, then S+T is assigned to B(J), then to A(I).

(4) There is an order of precedence in performing operations. For example, in A+B*C, the multiplication B*C is done first and then A is added to the result. The reason is that multiplication has a higher precedence than addition. The order of precedence (highest to lowest) is as follows:

```
subscripting
function evaluation
user-defined operators
      %          (precision operation)
      ↑ or **   (exponentiation)
  * / DIV MOD   (multiplying and dividing operations)
      + -       (addition and subtraction)
= <>  GEQ LEQ NEQ   (relational operations)
      NOT
      AND
    OR XOR
←=    ← := ::=   (assignment operations)
      :
      ,
      ;
```

The exchange operation (== or :=:) has lower precedence than subscripting and higher precedence than semicolon (;), but does not have a precedence relative to other operations, since no other operations may appear on either side of it.

TO ALL MANUAL HOLDERS:

Appendices D. and E. are not included in this OSCAR Manual, but will follow later. They should be added on to the back of this manual.

March 20, 1968
Revised July 15, 1968
Revised November 22, 1968
Revised September 25, 1969

APPENDIX F: Commands in OSCAR

In the following descriptions, (lun) denotes a logical unit number (a decimal integer from 0 to 99). (lunlist) denotes a list of logical unit numbers, separated by spaces or commas. In general, upper case letters are fixed (to be typed as is), while lower case letters denote variable quantities.

Part I - User Commands

&ASCII, c1, c2, ...

Print ASCII codes c1, c2, ... on the teletype (codes expressed in octal).

&ASCII, v1, v2 ...

If v1, v2, ... are simple variables whose values are non-negative integers, prints their values as ASCII codes on the teletype.

&AUTODUMP,lun

Release (lun) and then write the user's storage area and all of OSCAR on (lun) in two records. If this is later AUTOLOADed, it will restore OSCAR to the condition at time of AUTODUMPing. If (lun) is omitted, logical unit 0 is used.

&AUTOLOAD, lun

Rewind (lun), read one record from it, and jump to location 0. If (lun) is omitted, logical unit 0 is used. This command is equivalent to the OS3 control statement #AUTOLOAD,lun.

&BKSP,lunlist

Each logical unit in the list is backspaced one record.

&BKSPACE,lunlist

Same as &BKSP.

&BLANK

When used at a CRT, this command causes the screen to be blanked (cleared).

&CONTROL,lun

Read OSCAR statements and/or commands from (lun). If end-of-data is reached, if a file mark is read, or if an error occurs, revert to teletype control. (An error also causes outputs and data inputs to revert to the teletype). A line containing only the word ESCAPE will cause mode change (switch from normal mode to command mode, or vice-versa).

&CONTROL,lun,TTY

Same as &CONTROL,lun except print on teletype the information read from (lun).

&CONTROL,TTY

Read OSCAR statements and/or commands from the teletype.

&DATE

Print out the current date and time.

&DELETE,lun=name

Delete the name from the file which is currently equipped to (lun).

&EQUIP,lun=name

Equip logical unit (lun) to the hardware type or saved file specified by (name).

&FP,lun

Protect the file which is equipped to (lun).

&FWDSPACE,lunlist

Each logical unit in the list is spaced forward one record.

&FWSP,lunlist

Same as &FWDSPACE.

&INPUT,lun

Read data inputs from (lun). (Data inputs are constants read in by READ, READCHAR, and READLINE statements). If end-of-data is reached, if a file mark is read, or if an error occurs, switch Control, Input, and Output to the teletype and print an error message.

&INPUT,lun,TTY

Same as &INPUT,lun except print on teletype information read from (lun).

&INPUT,TTY

Read data inputs from teletype.

&OUTPUT,lun

Write all following outputs on (lun), until another &OUTPUT command is given. If an error occurs, revert to teletype output. (An error also causes control and data inputs to revert to the teletype.)

&OUTPUT,lun,TTY

Write following outputs on (lun), and also on the teletype.

&OUTPUT,TTY

Write following outputs on teletype.

&PARTLIST

Print a list of all part numbers that currently exist.

&PRECISION,<integer>

Standard precision is changed to <integer>; initially it is 6.

&PROGDUMP

Print all stored program parts that exist.

&PROGRAM, part no., interval

Accept stored program as input. Print a step number, allow a statement to be typed in, add the interval and print the next

step number, etc. The interval must be less than 1. If it is omitted, an interval of .01 is used. The part number may be an integer, or an integer with a fraction part, denoting the initial step number. If both interval and part number are omitted, the part number is assumed to be 1 and the interval is .01. Use [escape] or [alt mode] to exit from &PROGRAM mode.

&RECORD ON lun

Effective only at a CRT. Causes all subsequent CRT inputs and outputs to be written on the specified (lun).

&RECORD OFF

Stops recording.

&RELEASE,lunlist

Release each logical unit in the list.

&RESTART

Restart OSCAR. Clear user's storage area, etc.

&REWIND,lunlist

Rewind each logical unit in the list.

&RFP, lun

Remove protection from the non-saved file equipped to (lun).

&RFP, lun = name

Remove protection from the saved file (name), which is currently equipped to (lun).

&SAVE,lun=name

Save the file currently equipped to (lun), under the name specified.

&SBPFM,lunlist

On each logical unit in the list, search backward past file mark.

&SEFB,lunlist

Same as &SBPFM. (above)

&SEFF,lunlist

Same as &SFPFM. (below)

&SFPFM,lunlist

On each logical unit in the list, search forward past file mark.

&STORAGE,n

Add n more blocks of storage to user's storage area, if possible. (A block contains 2048 words,) If n is omitted, 1 is assumed.

&TIME

Print out (in seconds) the amount of computer time used since logging on.

&UDUMP,lun

Release (lun) and then write one record containing the user's storage area. If this is later ULOADed by the same version of OSCAR under which it was UDUMPed, it will restore conditions at time of UDUMPing. If (lun) is omitted, logical unit 0 is used.

&ULOAD,lun

Rewind (lun) and read one record into user's storage area. This record must have been UDUMPed by the same version of OSCAR as the one being used, or unpredictable effects will occur. If (lun) is omitted, logical unit 0 is used.

&UNEQUIP, lun

Unequip the file or device which is equipped to (lun).

&WEOF,lunlist

Same as &WFM. (below)

&WFM,lunlist

Write a file mark on each logical unit in the list.

&WIDTH, integer

Set the console line width to the number of characters specified (decimal integer). WIDTH is normally 72 for a teletype, 50 for a CRT, 132 in a batch job.

Part II

Commands of Use Mainly to System Personnel

&DEBUG ON

Turn on DBFLAG. Causes OSCAR to enter "command" mode whenever an error occurs. Allows one to get dumps, etc., before error conditions are "cleaned up."

&DEBUG ON lun

Same as DEBUG ON, except reads commands from specified (lun) if an error occurs.

&DEBUG OFF

Turn off DBFLAG. (DBFLAG is normally off.)

&DUMP

Print in octal all non-zero words in user's storage area.

&DUMP fwa,lwa

Print in octal the non-zero words in memory, from (fwa) to (lwa), where (fwa) and (lwa) are octal addresses.

&ERRPRINT

Print the last set of error messages that has been stacked. (If MI is used, error messages are stacked but not printed. This command enables one to find out what the messages are.)

&EXECUTE ON

Turn on RUNFLAG. Execute OSCAR statements. (RUNFLAG is normally on.)

&EXECUTE OFF

Turn off RUNFLAG. Do not execute OSCAR statements.

&FREELIST

Print the free storage lists.

&FREELIST,n1,n2,...

Print the free storage lists for blocks of sizes (n1), (n2), etc. (The sizes are expressed as decimal numbers.)

&GO

Valid only in OSCAR command mode (see &DEBUG). Clean up error conditions and go back to normal mode.

&MAP

Print a list of subprograms in OSCAR, with their initial addresses (in octal).

&MAP name1, name2, ...

Print the addresses of the subprograms whose names are given.

&MAP addr1, addr2, ...

Determine which subprograms contain the given octal addresses and prir. their names and addresses.

&OCTOUT var1, var2, ...

Print in octal the values of the specified variables. (non-subscripted variables, only).

&OCTOUT str1, str2, ...

Print in octal the values pointed to by the entries in the symbol table at relative locations (str1), (str2), etc., where the (str)'s are small octal integers (less then 4000_8).

&OCTOUT ptr1, ptr2, ...

Print in octal the values pointed to by the pointers (ptr1), (ptr2), etc., where the pointers are expressed in octal.

&POLISH ON

Turn on PSFLAG. Print out the Polish string representation of each OSCAR statement read in.

&POLISH OFF

Turn off PSFLAG. Do not print Polish strings. (PSFLAG is normally off.)

&RENAME, v1 = v2

If variable v1 is in the symbol table change its name to v2.

&RUBOUT

Release values pointed to by symbol table, clear symbol table, free storage list, and pad, dump what is left, and restart.

&SYMTAB

Print the symbol table.

&TEMPOUT ON

Turn on OCTLIST flag. Print, in octal, temporary structures, such as subarrays.

&TEMPOUT OFF

Turn off OCTLIST flag. Do not print temporary structures. (OCTLIST flag is normally off.)